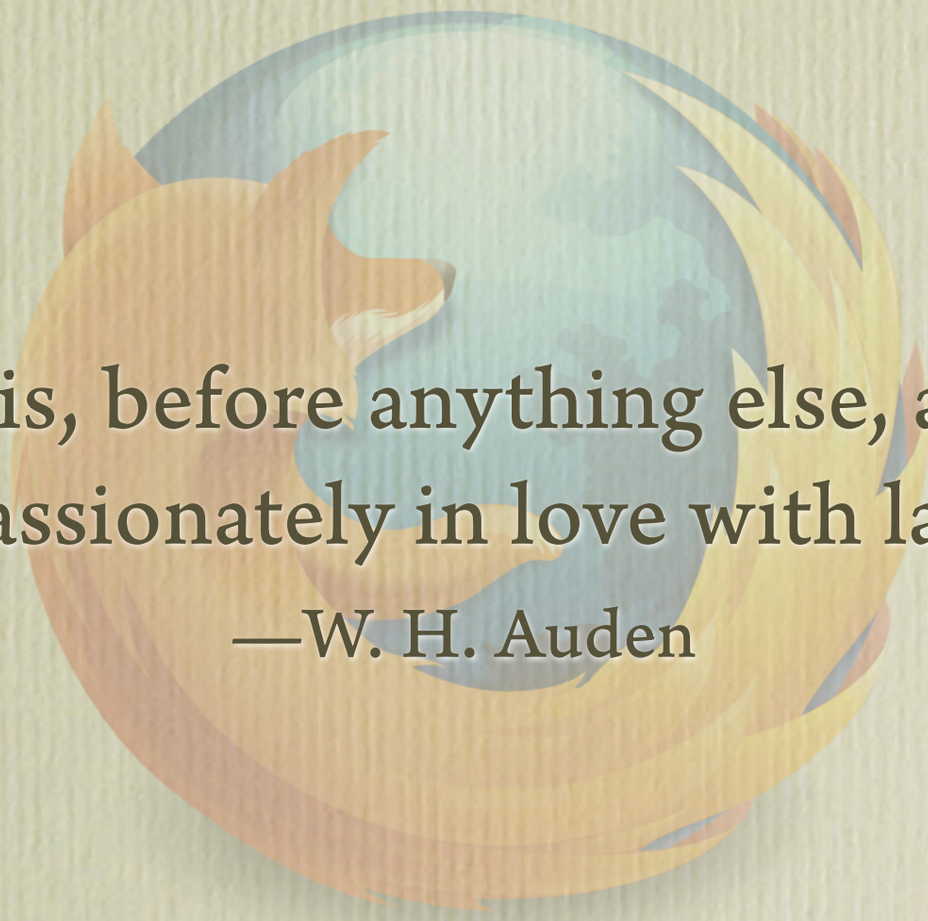
Designing Poetic APIs

by Erik Rose

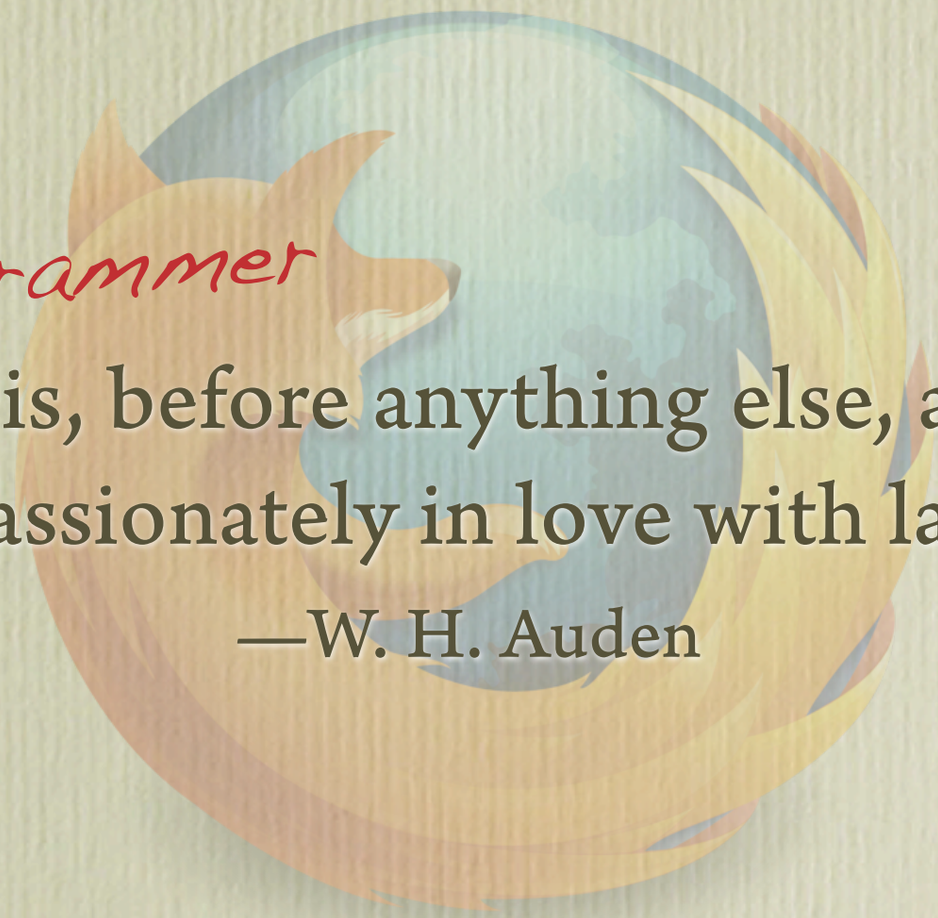


"A poet is, before anything else, a person who is passionately in love with language."

—W. H. Auden

Designing Poetic APIs

by Erik Rose



programmer

"A ~~poet~~ is, before anything else, a person who is passionately in love with language."

—W. H. Auden

Sapir-Whorf

"The limits of my language are the limits of my world."

—Ludwig Wittgenstein

Intellectual Intelligibility

"Any fool can write code that a computer can understand.
Good programmers write code that humans can understand."

—Martin Fowler

```
req = urllib2.Request(gh_url)
password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
password_manager.add_password(
    None, 'https://api.github.com', 'user', 'pass')
auth_manager = urllib2.HTTPBasicAuthHandler(password_manager)
opener = urllib2.build_opener(auth_manager)
urllib2.install_opener(opener)
handler = urllib2.urlopen(req)
print handler.getcode()
```

```
req = urllib2.Request(gh_url)
password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
password_manager.add_password(
    None, 'https://api.github.com', 'user', 'pass')
auth_manager = urllib2.HTTPBasicAuthHandler(password_manager)
opener = urllib2.build_opener(auth_manager)
urllib2.install_opener(opener)
handler = urllib2.urlopen(req)
print handler.getcode()
```

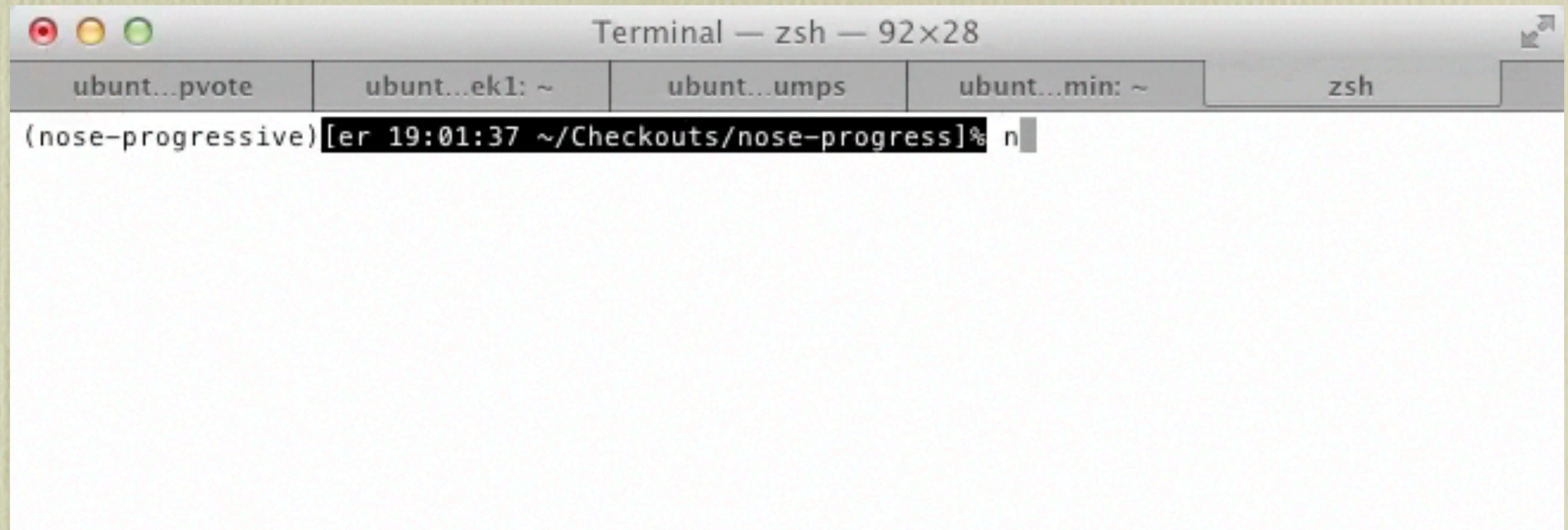
```
r = requests.get('https://api.github.com', auth=('user', 'pass'))
print r.status_code
```

Don't Be An Architecture Astronaut

"It's hard to make predictions, especially about the future."

—Robert Storm Petersen

The best libraries are extracted, not invented.


Terminal — zsh — 92×28
ubunt...pvote
ubunt...ek1: ~
ubunt...umps
ubunt...min: ~
zsh
(nose-progressive)[er 19:01:37 ~/Checkouts/nose-progress]% n

```
def terminal_codes(self, stream):
    capabilities = ['bold', 'sc', 'rc', 'sgr0', 'el']
    if hasattr(stream, 'fileno') and isatty(stream.fileno()):
        # Explicit args make setupterm() work even when -s is passed:
        setupterm(None, stream.fileno()) # so things like tigetstr() work
        codes = dict((x, tigetstr(x)) for x in capabilities)
        cup = tigetstr('cup')
        codes['cup'] = lambda line, column: tparm(cup, line, column)
    else:
        # If you're crazy enough to pipe this to a file or something,
        # don't output terminal codes:
        codes = defaultdict(lambda: '', cup=lambda line, column: '')
    return codes

...
self._codes = terminal_codes(stdout)
...

class AtLine(object):
    def __enter__(self):
        """Save position and move to progress bar, col 1."""
        self.stream.write(self._codes['sc']) # save position
        self.stream.write(self._codes['cup'](self.line, 0))

    def __exit__(self, type, value, tb):
        self.stream.write(self._codes['rc']) # restore position

...
print self._codes['bold'] + results + self._codes['sgr0']
```

```
def terminal_codes(self, stream):
    capabilities = ['bold', 'sc', 'rc', 'sgr0', 'el']
    if hasattr(stream, 'fileno') and isatty(stream.fileno()):
        # Explicit args make setupterm() work even when -s is passed:
        setupterm(None, stream.fileno()) # so things like tigetstr() work
        codes = dict((x, tigetstr(x)) for x in capabilities)
        cup = tigetstr('cup')
        codes['cup'] = lambda line, column: tparm(cup, line, column)
    else:
        # If you're crazy enough to pipe this to a file or something,
        # don't output terminal codes:
        codes = defaultdict(lambda: '', cup=lambda line, column: '')
    return codes

...
self._codes = terminal_codes(stdout)
...

class AtLine(object):
    def __enter__(self):
        """Save position and move to progress bar, col 1."""
        self.stream.write(self._codes['sc']) # save position
        self.stream.write(self._codes['cup'](self.line, 0))

    def __exit__(self, type, value, tb):
        self.stream.write(self._codes['rc']) # restore position

...
print self._codes['bold'] + results + self._codes['sgr0']
```

Tasks

Tasks

Print some text with formatting.

Tasks

Print some text with formatting.

Print some text at a location, then snap back.

Language Constructs

Language Constructs

Functions, arguments, keyword arguments

Language Constructs

Functions, arguments, keyword arguments

Decorators

Language Constructs

Functions, arguments, keyword arguments

Decorators

Context managers

Language Constructs

Functions, arguments, keyword arguments

Decorators

Context managers

Classes (really more of an implementation detail)

Language Constructs

Functions, arguments, keyword arguments

Decorators

Context managers

Classes (really more of an implementation detail)

Patterns, Protocols, Interfaces, and Conventions

Language Constructs

Functions, arguments, keyword arguments

Decorators

Context managers

Classes (really more of an implementation detail)

Patterns, Protocols, Interfaces, and Conventions

Sequences

Language Constructs

Functions, arguments, keyword arguments

Decorators

Context managers

Classes (really more of an implementation detail)

Patterns, Protocols, Interfaces, and Conventions

Sequences

Iterators

Language Constructs

Functions, arguments, keyword arguments

Decorators

Context managers

Classes (really more of an implementation detail)

Patterns, Protocols, Interfaces, and Conventions

Sequences

Iterators

Mappings

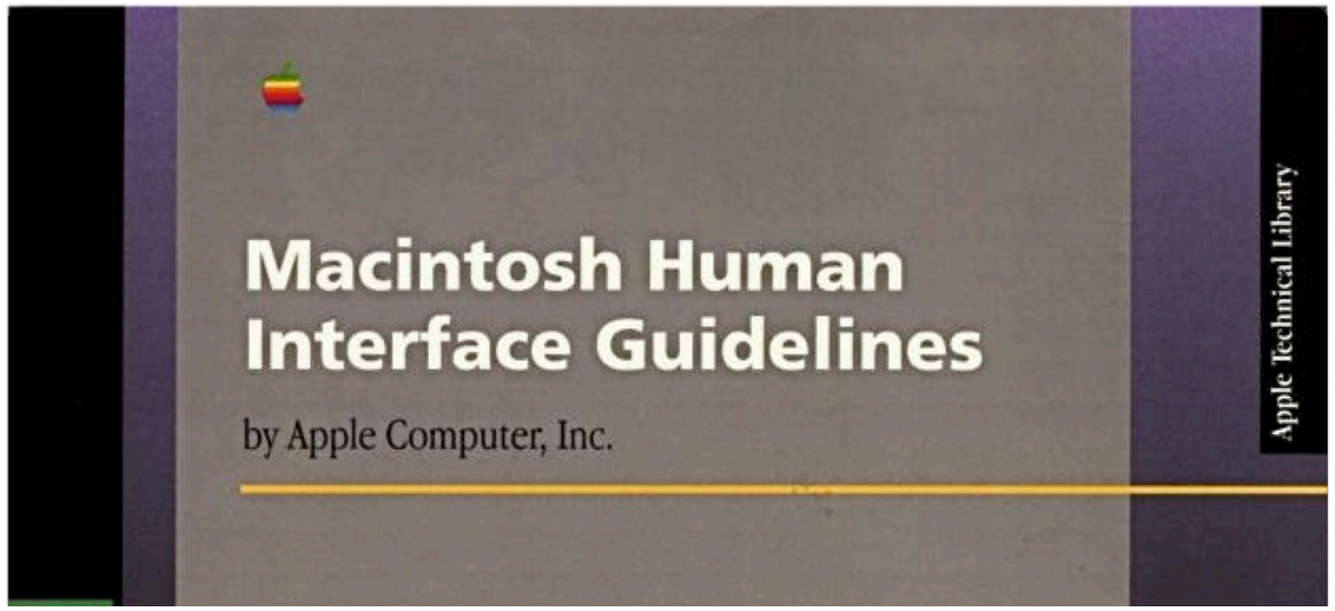
Consistency

"Think like a wise man, but communicate
in the language of the people."
—William Butler Yeats

Macintosh Human Interface Guidelines

by Apple Computer, Inc.

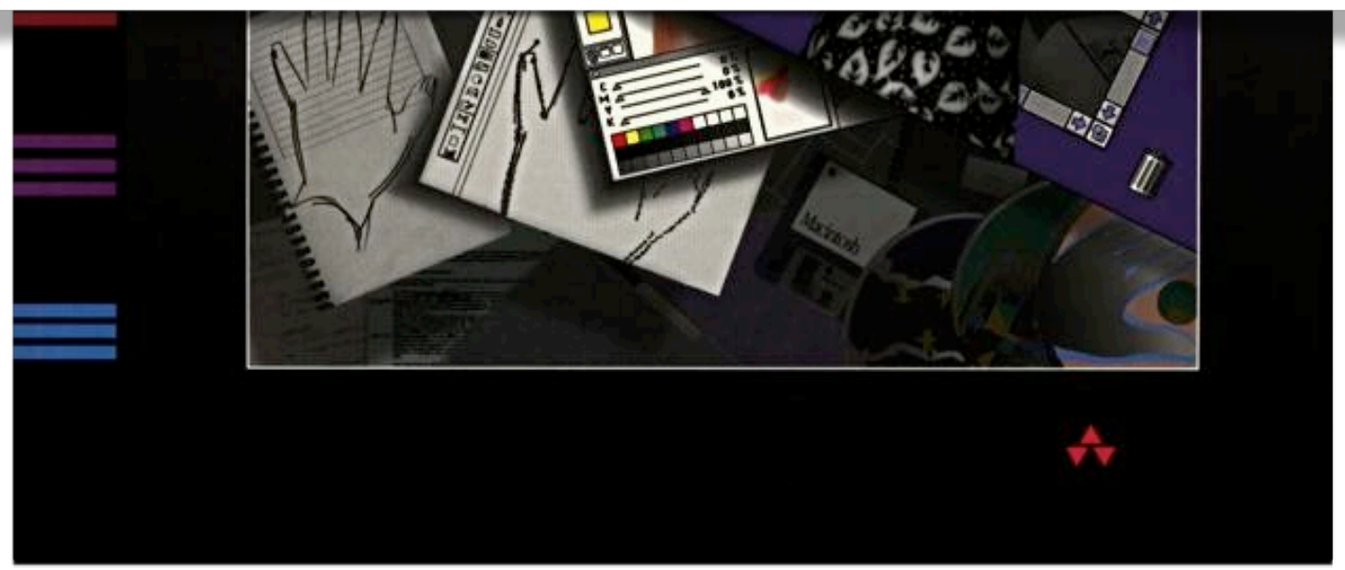
Apple Technical Library



`get(key, default)`

is better than

`fetch(default, key)`



Tasks

Print some text at a location, then snap back.

Print some text with formatting.

Tasks

Print some text at a location, then snap back.

Print some text with formatting.

```
print_at('Hello world', 10, 2)
```

```
with location(10, 2):
    print 'Hello world'
    for thing in range(10):
        print thing
```

Tasks

Print some text at a location, then snap back.

Print some text with formatting.

```
print_formatted('Hello world', 'red', 'bold')
print color('Hello world', 'red', 'bold')
print color('<red><bold>Hello world</bold></red>')
print red(bold('Hello') + 'world')

print codes['red'] + codes['bold'] + \
      'Hello world' + codes['normal']

print '{t.red}Hi{t.bold}Mom{t.normal}'.format(t=terminal)
print terminal.red_bold + 'Hello world' + terminal.normal
```

Tasks

Print some text at a location, then snap back.

Print some text with formatting.

```
print_formatted('Hello world', 'red', 'bold')
print color('Hello world', 'red', 'bold')
print color('<red><bold>Hello world</bold></red>')
print red(bold('Hello') + 'world')

print codes['red'] + codes['bold'] + \
      'Hello world' + codes['normal']

print '{t.red}Hi{t.bold}Mom{t.normal}'.format(t=terminal)
print terminal.red_bold + 'Hello world' + terminal.normal
```

Tasks

Print some text at a location, then snap back.

Print some text with formatting.

```
print_formatted('Hello world', 'red', 'bold')
print color('Hello world', 'red', 'bold')
print color('<red><bold>Hello world</bold></red>')
print red(bold('Hello') + 'world')

print codes['red'] + codes['bold'] + \
      'Hello world' + codes['normal']

print '{t.red}Hi{t.bold}Mom{t.normal}'.format(t=terminal)
print terminal.red_bold + 'Hello world' + terminal.normal
```

Tasks

Print some text at a location, then snap back.

Print some text with formatting.

```
print_formatted('Hello world', 'red', 'bold')
print color('Hello world', 'red', 'bold')
print color('<red><bold>Hello world</bold></red>')
print red(bold('Hello') + 'world')

print codes['red'] + codes['bold'] + \
      'Hello world' + codes['normal']

print '{t.red}Hi{t.bold}Mom{t.normal}'.format(t=terminal)
print terminal.red_bold + 'Hello world' + terminal.normal
```

```
from blessings import Terminal

t = Terminal()

print t.red_bold + 'Hello world' + t.normal
print t.red_on_white + 'Hello world' + t.normal
print t.underline_red_on_green + 'Hello world' + t.normal
```

```python
from blessings import Terminal

t = Terminal()

print t.red_bold + 'Hello world' + t.normal
print t.red_on_white + 'Hello world' + t.normal
print t.underline_red_on_green + 'Hello world' + t.normal
```

```
Article.objects.filter(tag__in=['color_red', 'color_blue'])
Article.objects.filter(tag__contains='color')
```

Consistency

warning signs

Consistency

warning signs

Frequent references to your docs or source

Consistency

warning signs

Frequent references to your docs or source

Feeling syntactically clever

Brevity

"The finest language is mostly made up
of simple, unimposing words."
—George Eliot

```
from blessings import Terminal
term = Terminal()

print term.bold + 'I am bold!' + term.normal
```

```
from blessings import Terminal
term = Terminal()

print term.bold + 'I am bold!' + term.normal

print term.bold('I am bold!')
```

```
from blessings import Terminal
term = Terminal()

print term.bold + 'I am bold!' + term.normal

print term.bold('I am bold!')
```

```
from blessings import Terminal
term = Terminal()

print term.bold + 'I am bold!' + term.normal

print term.bold('I am bold!')

print '{t.bold}Very {t.red}emphasized{t.normal}'.format(t=term)
```

Brevity

warning signs

Brevity

warning signs

Copying and pasting when writing against your API

Brevity

warning signs

Copying and pasting when writing against your API

Typing something irrelevant while grumbling
“Why can’t it just assume the obvious thing?”

Brevity

warning signs

Copying and pasting when writing against your API

Typing something irrelevant while grumbling
“Why can’t it just assume the obvious thing?”

Long arg lists, suggesting a lack of sane defaults

Composability

"Perfection is achieved not when there is nothing left to add
but when there is nothing left to take away."

—Antoine de Saint-Exupery

```
print_formatted('Hello world', 'red', 'bold')
```

```
print_formatted('Hello world', 'red', 'bold')

print_formatted('Hello world', 'red', 'bold', out=some_file)
```

```
print_formatted('Hello world', 'red', 'bold')

print_formatted('Hello world', 'red', 'bold', out=some_file)

print formatted('Hello world', 'red', 'bold')
```

Composabilty

warning signs

Composabilty

warning signs

Classes with lots of state

Composabilty

warning signs

Classes with lots of state

```
class ElasticSearch(object):
    """
    An object which manages connections to elasticsearch and acts as a
    go-between for API calls to it"""

    def index(self, index, doc_type, doc, id=None, force_insert=False,
              query_params=None):
        """Put a typed JSON document into a specific index to make it
        searchable."""

    def search(self, query, **kwargs):
        """Execute a search query against one or more indices and get back search
        hits."""

    def more_like_this(self, index, doc_type, id, mlt_fields, body='', query_params=None):
        """Execute a "more like this" search query against one or more fields and
        get back search hits."""

    . . .
```

Composabilty

warning signs

Classes with lots of state

```
class ElasticSearch(object):
    """
    An object which manages c
    go-between for API calls

    def index(self, index, do
              query_params=No
        """Put a typed JSON d
        searchable."""

    def search(self, query, *
        """Execute a search q
        hits."""

    def more_like_this(self,
        """Execute a "more li
        get back search hits.

    . . .
```

```
class PenaltyBox(object):
    """A thread-safe bucket of servers (or other things) that may have
    downtime."""

    def get(self):
        """Return a random server and a bool indicating whether it was from the
        dead list."""

    def mark_dead(self, server):
        """Guarantee that this server won't be returned again until a period of
        time has passed, unless all servers are dead."""

    def mark_live(self, server):
        """Move a server from the dead list to the live one."""
```

Composabilty

warning signs

Classes with lots of state

Deep inheritance hierarchies

Composabilty

warning signs

Classes with lots of state

Deep inheritance hierarchies

Violations of the Law of Demeter

Composabilty

warning signs

Classes with lots of state

Deep inheritance hierarchies

Violations of the Law of Demeter

Mocking in tests

Composabilty

warning signs

Classes with lots of state

Deep inheritance hierarchies

Violations of the Law of Demeter

Mocking in tests

```
print_formatted('Hello world', 'red', 'bold')
```

Composabilty

warning signs

Classes with lots of state

Deep inheritance hierarchies

Violations of the Law of Demeter

Mocking in tests

```
print_formatted('Hello world', 'red', 'bold')

print formatted('Hello world', 'red', 'bold')
```

Composabilty

warning signs

Classes with lots of state

Deep inheritance hierarchies

Violations of the Law of Demeter

Mocking in tests

Options

Plain Data

"All the great things are simple, and many
can be expressed in a single word…"

—Sir Winston Churchill

`RawConfigParser.`**`options`**`(`*section*`)`

Returns a list of options available in the specified *section*.

`RawConfigParser.`**`has_option`**`(`*section*, *option*`)`

If the given section exists, and contains the given option, return `True`; otherwise return `False`.

`RawConfigParser.`**`get`**`(`*section*, *option*`)`

Get an *option* value for the named *section*.

`RawConfigParser.`**`options`**`(`*section*`)`

Returns a list of options availa[illegible]e in the [illegible]ecified *section*.

`RawConfigParser.`**`has_opti`**[illegible]*tion*`)`

If the given section exists, and [illegible]he given option, return **`True`**; otherwise return **`False`**.

`RawConfigParser.`**`get`**`(`*section*, *option*`)`

Get an *option* value for the named *section*.

`RawConfigParser.`**`options`**`(`*section*`)`

Returns a list of options available in the specified *section*.

`RawConfigParser.`**`has_option`**`(`*section*`,` *option*`)`

If the given section exists, and contains the given option, return **`True`**; otherwise return **`False`**.

`RawConfigParser.`**`get`**`(`*section*`,` *option*`)`

Get an *option* value for the named *section*.

`RawConfigParser.`**`read`**`(`*filename*`)`

`RawConfigParser.`**`options`**`(`*section*`)`

Returns a list of options available in the specified *section*.

`RawConfigParser.`**`has_option`**`(`*section*`,` *option*`)`

If the given section exists, and contains the given option, return **`True`**; otherwise return **`False`**.

`RawConfigParser.`**`get`**`(`*section*`,` *option*`)`

Get an *option* value for the named *section*.

`RawConfigParser.`**`read`**`(`*filename*`)`

`RawConfigParser.`**`options`**`(`*section*`)`

Returns a list of options availa e in the ecified *section*.

`RawConfigParser.`**`has_opti`**`tion)`

If the given section exists, and he given option, return **`True`**; otherwise return **`False`**.

`RawConfigParser.`**`get`**`(`*section*, *ption*`)`

Get an *option* value for the named *section*.

`RawConfigParser.`**`read`**`(`*filename*`)`

`RawConfigParser.`**`parse`**`(`*string*`)`

`RawConfigParser.`**`options`**`(`*section*`)`

Returns a list of options availa e in the ecified *section*.

`RawConfigParser.`**`has_opti`** tion)

If the given section exists, and he given option, return **`True`**; otherwise return **`False`**.

`RawConfigParser.`**`get`**`(`*section*, *option*`)`

Get an *option* value for the named *section*.

`RawConfigParser.`**`read`**`(`*filename*`)`

```
RawConfigParser.parse(string)

charming_parser.parse(file_contents('some_file.ini'))
```

```
class Node(dict):
    """A wrapper around a native Reflect.parse dict
    providing some convenience  methods and some caching
    of expensive computations"""
```

```
class Node(dict):
    """A wrapper around a native Reflect.parse dict
    providing some convenience  methods and some caching
    of expensive computations"""

    def walk_up(self):
        """Yield each node from here to the root of the
        tree, starting with myself."""
        node = self
        while node:
            yield node
            node = node.get('_parent')
```

```
class Node(dict):
    """A wrapper around a native Reflect.parse dict
    providing some convenience  methods and some caching
    of expensive computations"""

    def walk_up(self):
        """Yield each node from here to the root of the
        tree, starting with myself."""
        node = self
        while node:
            yield node
            node = node.get('_parent')

def nearest_scope_holder(self):
    """Return the nearest node that can have its own
    scope, potentially including myself.

    This will be either a FunctionDeclaration or a
    Program (for now).

    """
    return first(n for n in self.walk_up() if
                 n['type'] in ['FunctionDeclaration',
                               'Program'])
```

Plain Data

warning signs

Plain Data

warning signs

Users immediately transforming your output to another format

Plain Data

warning signs

Users immediately transforming your output to another format

Instantiating one object just to pass it to another

Plain Data

warning signs

Users immediately transforming your output to another format

Instantiating one object just to pass it to another

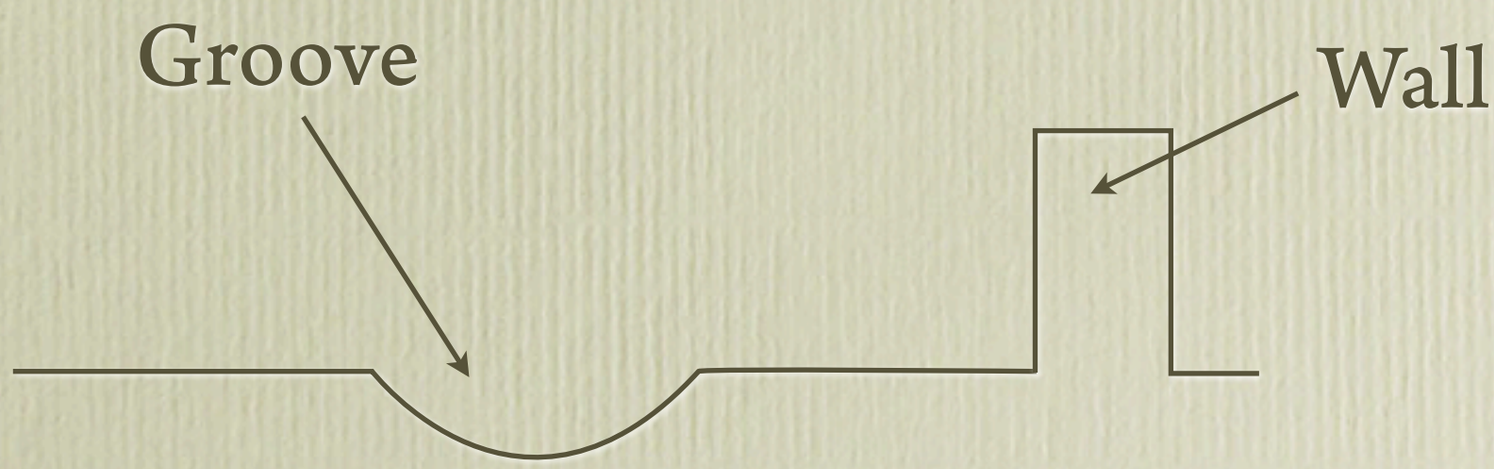
Rewriting language-provided things

Grooviness

"The bad teacher's words fall on his pupils like harsh rain;
the good teacher's, as gently as dew."

—Talmud: Ta'anith 7b

Grooviness



Avoid nonsense representations.

Avoid nonsense representations.

```
{
    "bool" : {
        "must" : {
            "term" : { "user" : "fred" }
        },
        "must_not" : {
            "range" : {
                "age" : { "from" : 12, "to" : 21 }
            }
        },
        "minimum_number_should_match" : 1,
        "boost" : 1.0
    }
}
```

Avoid nonsense representations.

```
{
    "bool" : {
        "must" : {
            "term" : { "user" : "fred" }
        },
        "must_not" : {
            "range" : {
                "age" : { "from" : 12, "to" : 21 }
            }
        },
        "minimum_number_should_match" : 1,
        "boost" : 1.0
    }
}
```

```
frob*ator AND snork
```

Old pyelasticsearch

```
def search(self, q=None, body=None, indexes=None, doc_types=None):
    """Execute an elasticsearch query, and return the results."""
```

Old pyelasticsearch

```
def search(self, q=None, body=None, indexes=None, doc_types=None):
    """Execute an elasticsearch query, and return the results."""

search(q='frob*ator AND snork', body={'some': 'query'})
```

Old pyelasticsearch

```
def search(self, q=None, body=None, indexes=None, doc_types=None):
    """Execute an elasticsearch query, and return the results."""

search(q='frob*ator AND snork', body={'some': 'query'})
search()
```

Old pyelasticsearch

```
def search(self, q=None, body=None, indexes=None, doc_types=None):
    """Execute an elasticsearch query, and return the results."""

s       q='frob*ator AND snork', body={'some': 'query'})
se      ()
```

Old pyelasticsearch

```
def search(self, q=None, body=None, indexes=None, doc_types=None):
    """Execute an elasticsearch query, and return the results."""

s      q='frob*ator AND snork', body={'some': 'query'})
se     ()
```

New pyelasticsearch

```
def search(self, query, indexes=None, doc_types=None):
    """Execute an elasticsearch query, and return the results."""
```

Old pyelasticsearch

```
def search(self, q=None, body=None, indexes=None, doc_types=None):
    """Execute an elasticsearch query, and return the results."""

s      q='frob*ator AND snork', body={'some': 'query'})
se     ()
```

New pyelasticsearch

```
def search(self, query, indexes=None, doc_types=None):
    """Execute an elasticsearch query, and return the results."""
```

Fail shallowly.

Resource acquisition is initialization.

Resource acquisition is initialization.

```
class PoppableBalloon(object):
    """A balloon you can pop"""

    def __init__(self):
        self.air = 0

    def fill(self, how_much):
        self.air = how_much
```

Resource acquisition is initialization.

```
class PoppableBalloon(object):
    """A balloon you can pop"""

    def __init__(self):
        self.air = 0

    def fill(self, how_much):
        self.air = how_much
```

```
class PoppableBalloon(object):
    """A balloon you can pop"""

    def __init__(self, initial_fill):
        self.air = initial_fill

    def fill(self, how_much):
        self.air = how_much
```

Compelling examples

Compelling examples

Grooviness

warning signs

Grooviness

warning signs

Representable nonsense

Grooviness

warning signs

Representable nonsense

Invariants that aren't

Grooviness

warning signs

Representable nonsense

Invariants that aren't

Lack of a clear starting point

Grooviness

warning signs

Representable nonsense

Invariants that aren't

Lack of a clear starting point

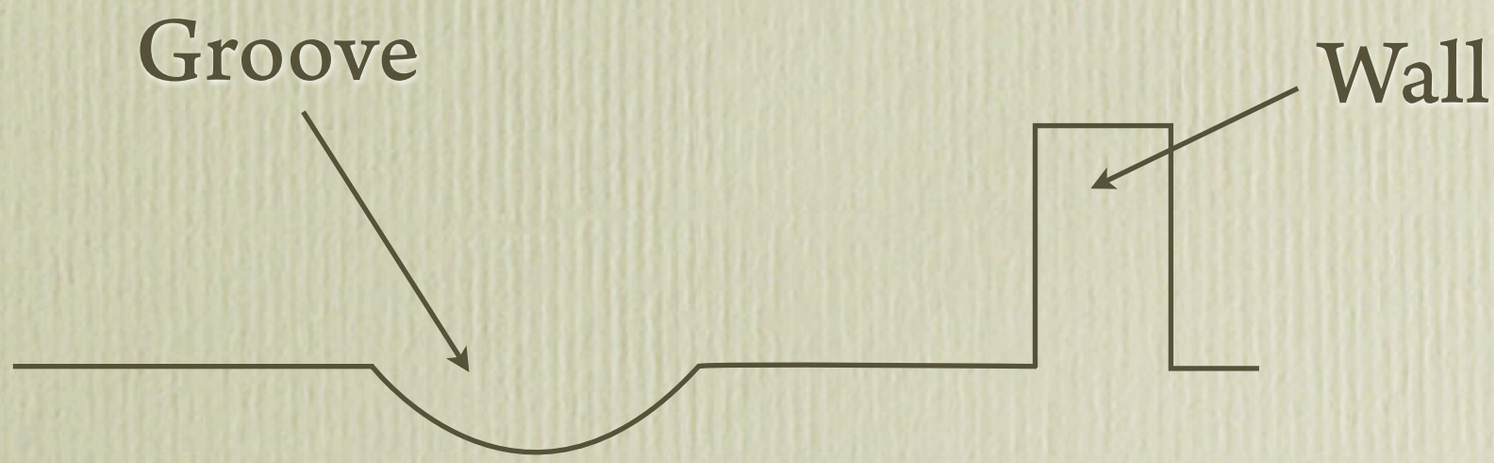
Long, complicated documentation

Safety

"I wish it was easier to hurt myself."

—Nobody, ever

Safety



```
update things set frob=2 where frob=1;
```

```
update things set frob=2 where frob=1;

update things set frob=2;
```

```
update things set frob=2 where frob=1;

update things set frob=2;

update things set frob=2 all;
```

```
update things set frob=2 where frob=1;

update things set frob=2;

update things set frob=2 all;
```

```
rm *.pyc
```

```
update things set frob=2 where frob=1;

update things set frob=2;

update things set frob=2 all;
```

```
rm *.pyc
rm *
```

```
update things set frob=2 where frob=1;

update things set frob=2;

update things set frob=2 all;
```

```
rm *.pyc
rm *
rm -f *
```

```
def delete(self, index, doc_type, id=None):
    """Delete a typed JSON document from a specific
    index based on its id."""
```

```
def delete(self, index, doc_type, id=None):
    """Delete a typed JSON document from a specific
    index based on its id."""
```

```
def delete(self, index, doc_type, id=None):
    """Delete a typed JSON document from a specific
    index based on its id."""
```

```
def delete(self, index, doc_type, id):
    """Delete a typed JSON document from a specific
    index based on its id."""

def delete_all(self, index, doc_type):
    """Delete all documents of the given doctype from
    an index."""
```

Exceptions > Return Values

Safety

warning signs

Safety

warning signs

Docs that say “remember to…” or “make sure you…”

Safety

warning signs

Docs that say “remember to…” or “make sure you…”
Surprisingly few—people will blame themselves.

Design Smell Checklist

Architecture Astronautics

- ✖ Inventing rather than extracting

Consistency

- ✖ Frequent references to your docs or source
- ✖ Feeling syntactically clever

Brevity

- ✖ Copying and pasting when writing against your API
- ✖ Grumbling about having to hand-hold the API

Plain Data

- ✖ Users immediately transforming your output to another format
- ✖ Rewriting language facilities
- ✖ Instantiating an object just to pass it

Composability

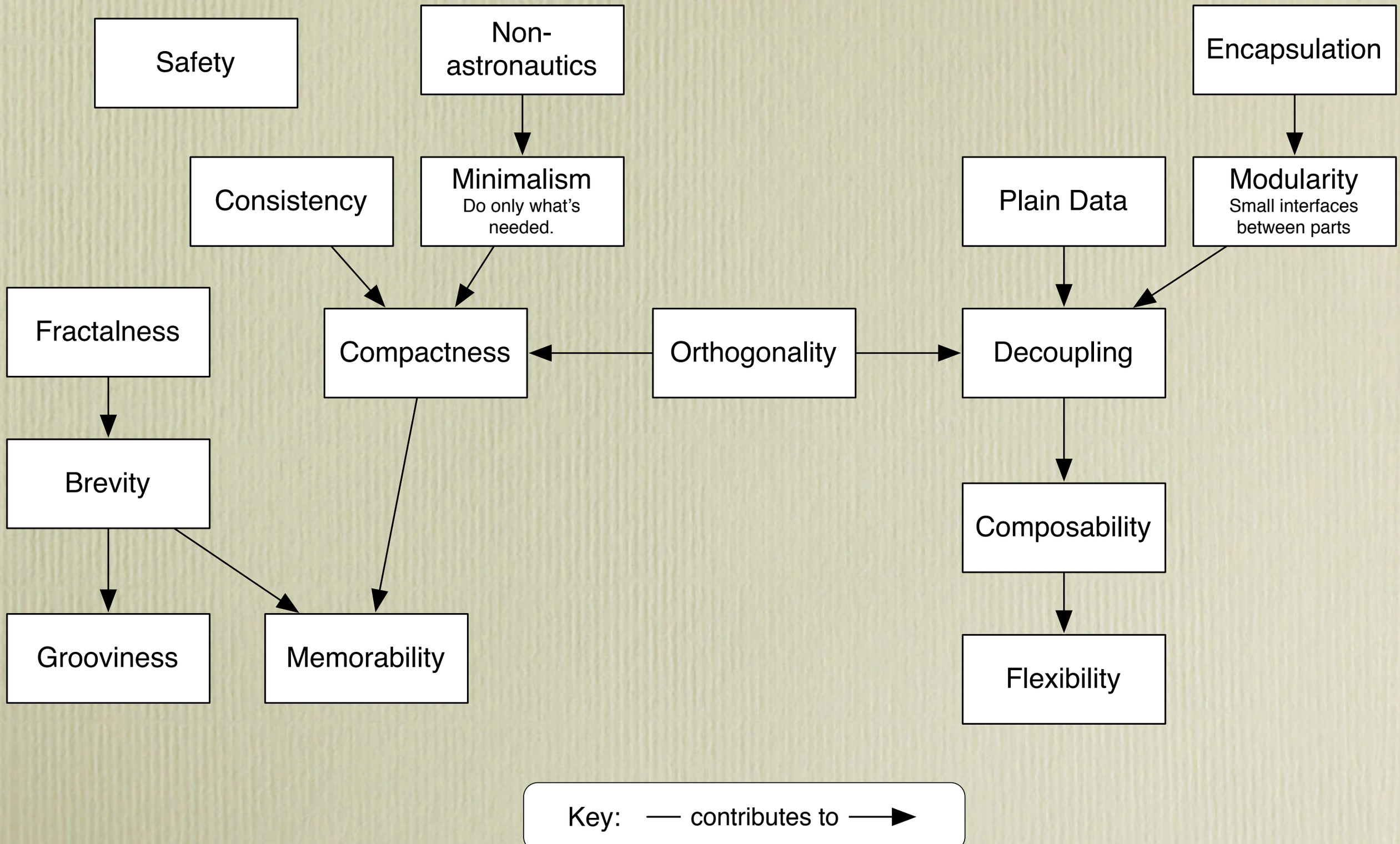
- ✖ Classes with lots of state
- ✖ Deep inheritance hierarchies
- ✖ Violations of the Law of Demeter
- ✖ Mocking in tests
- ✖ Bolted-on options

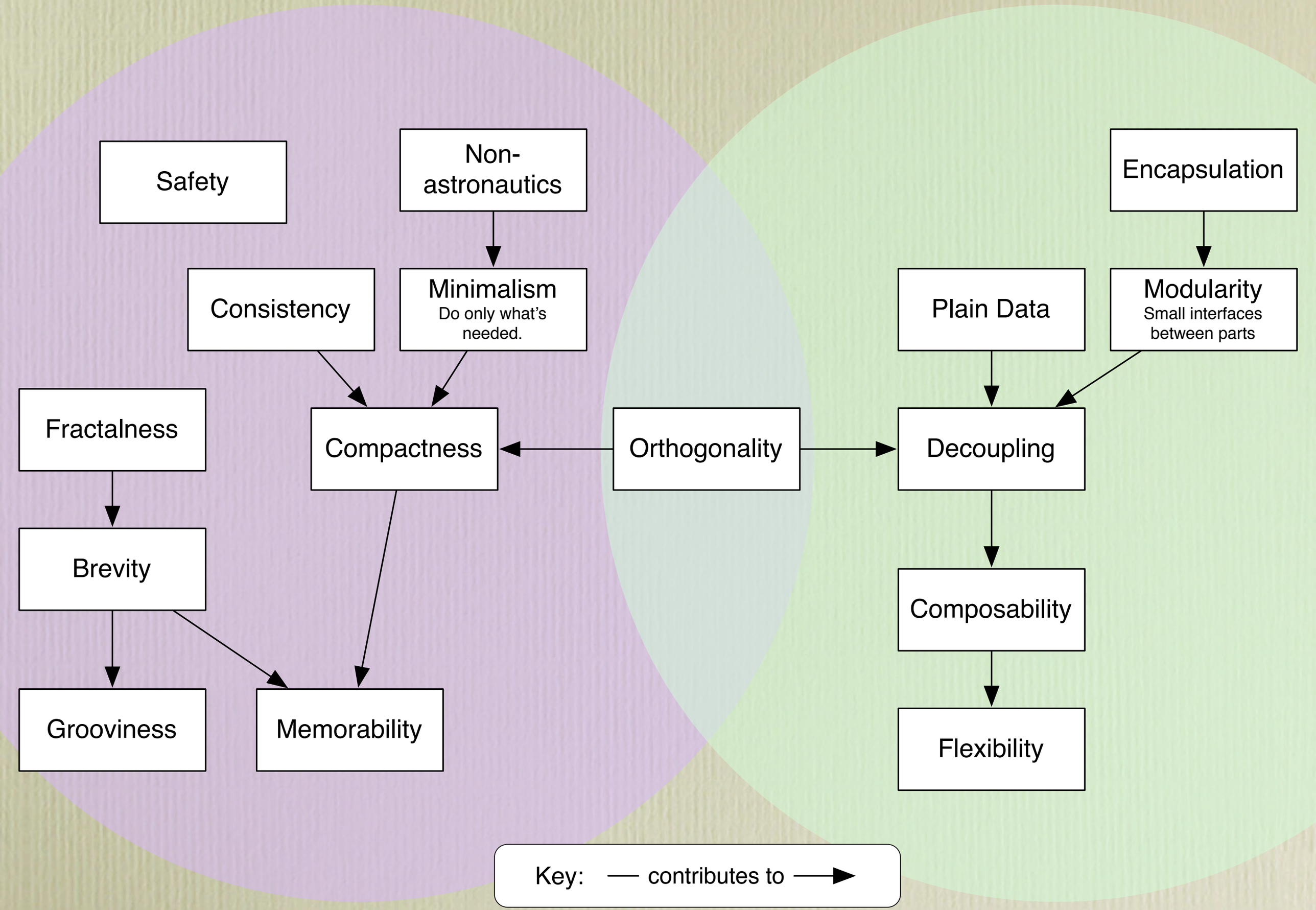
Grooviness

- ✖ Nonsense states
- ✖ Invariants that aren't
- ✖ Lack of a clear introductory path
- ✖ Complicated documentation

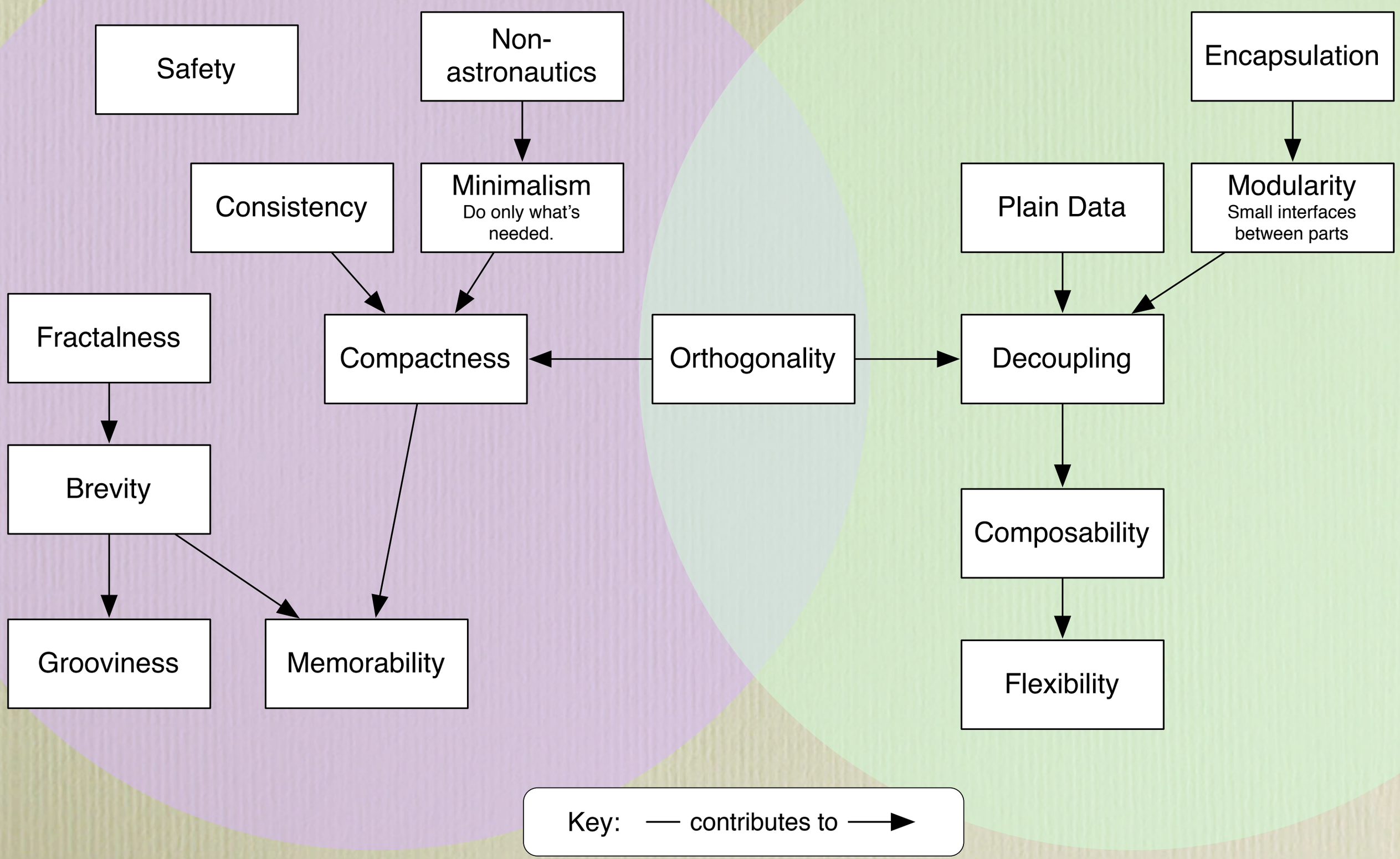
Safety

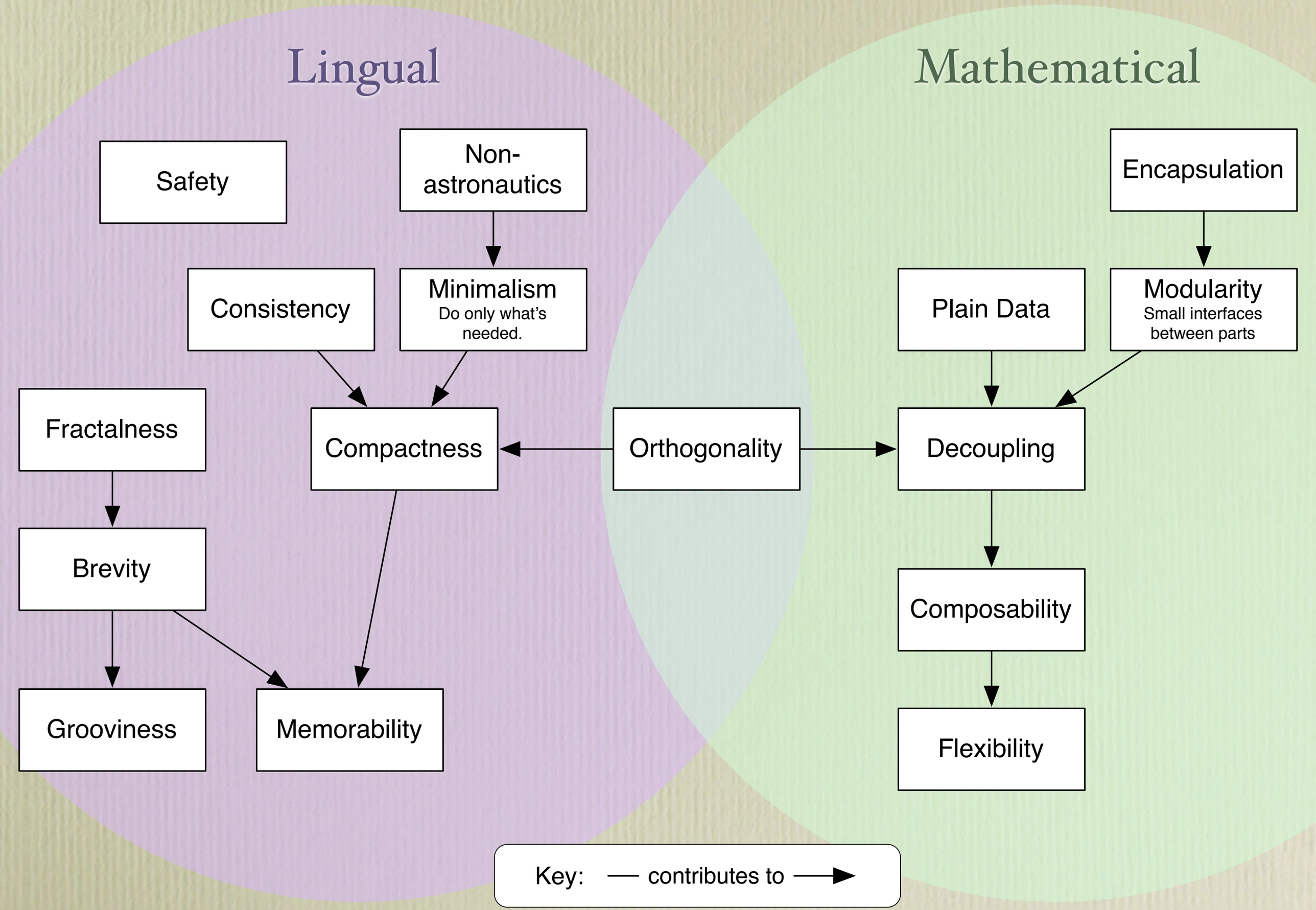
- ✖ Docs that say "make sure you…"
- ✖ Destructive actions without walls

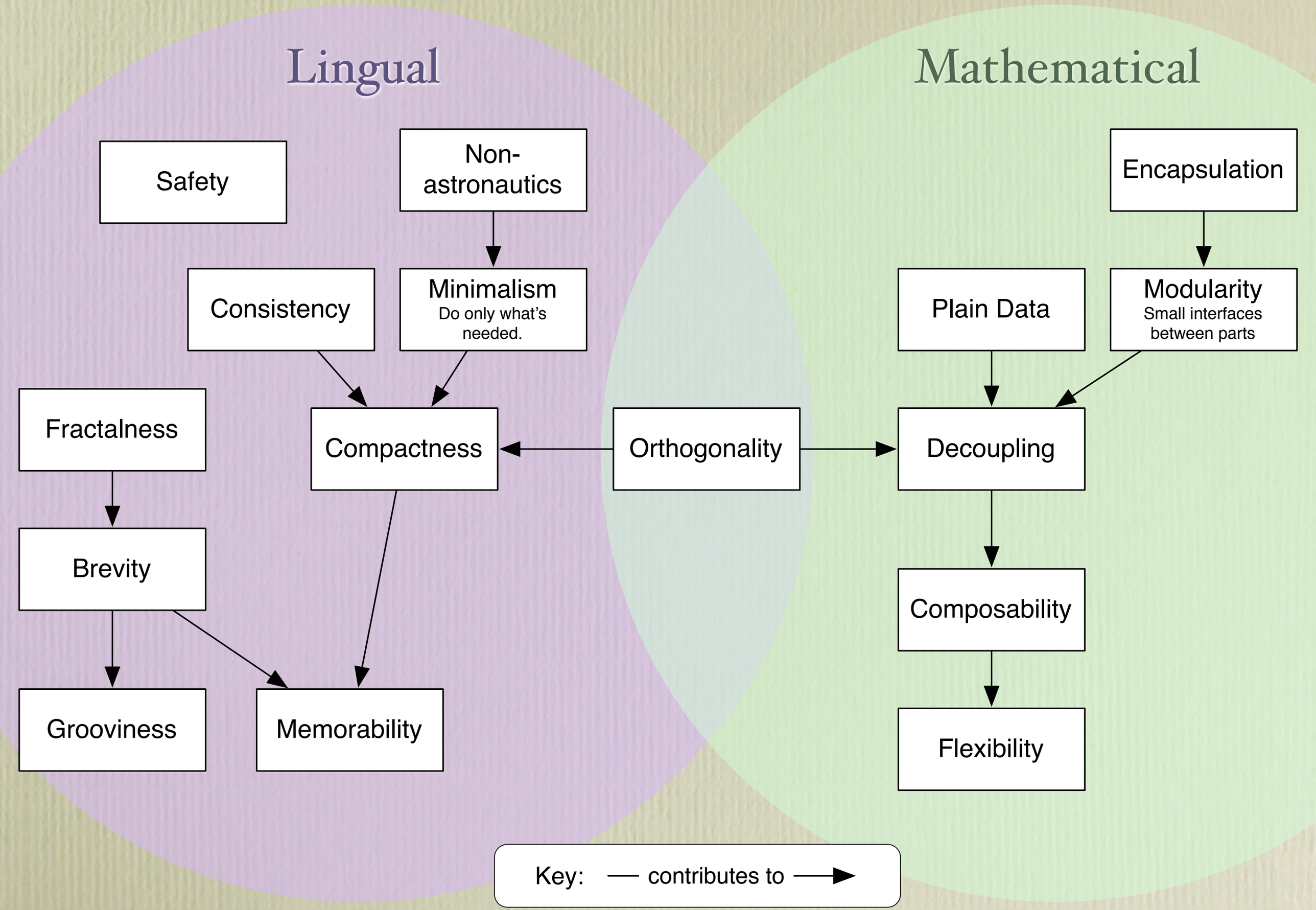

Safety
Non-
astronautics
Encapsulation
Consistency
Minimalism
Do only what's
needed.
Plain Data
Modularity
Small interfaces
between parts
Fractalness
Compactness
Orthogonality
Decoupling
Brevity
Composability
Grooviness
Memorability
Flexibility
Key: — contributes to ⟶

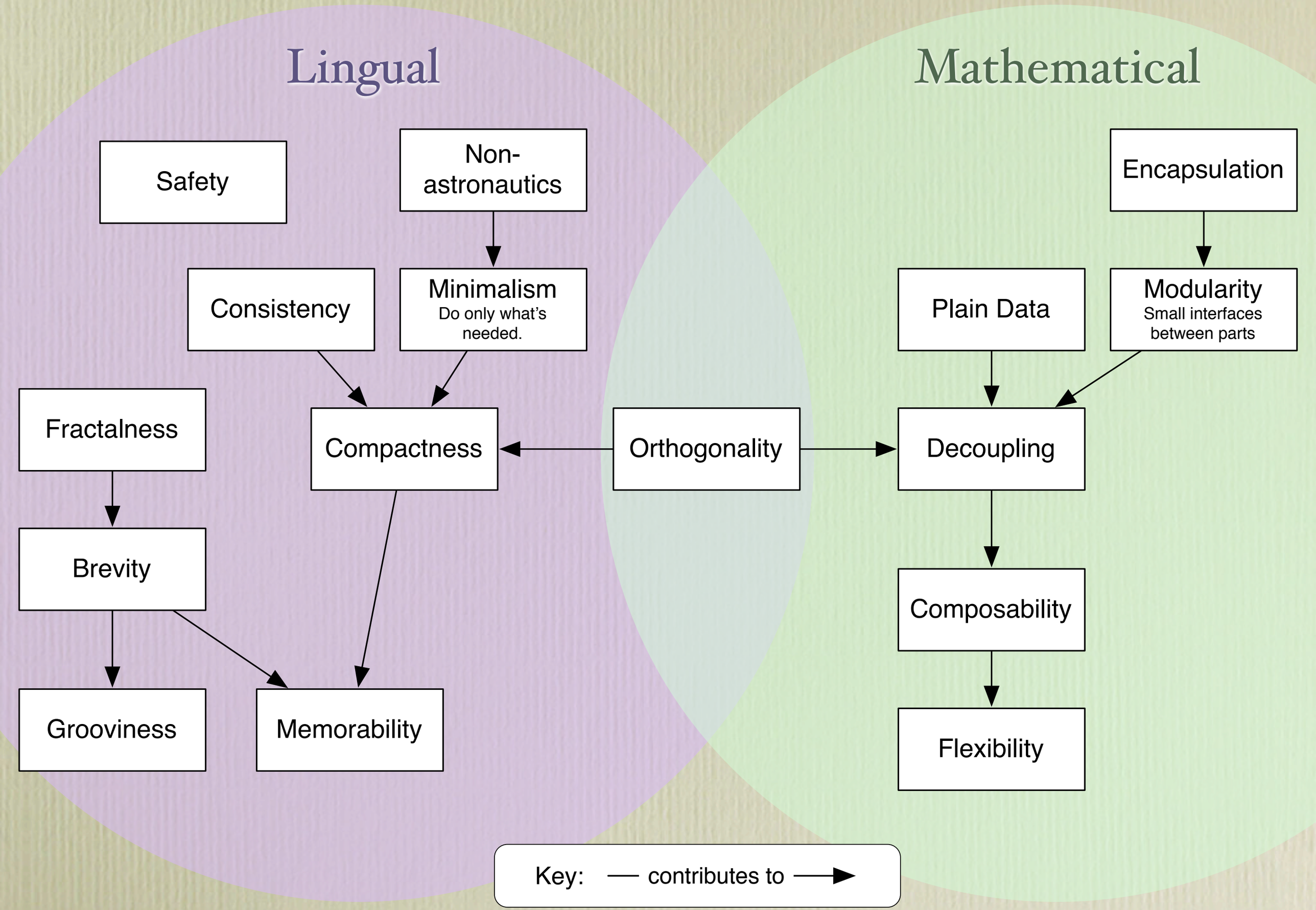

Safety
Non-
astronautics
Encapsulation
Consistency
Minimalism
Do only what's
needed.
Plain Data
Modularity
Small interfaces
between parts
Fractalness
Compactness
Orthogonality
Decoupling
Brevity
Composability
Grooviness
Memorability
Flexibility
Key: — contributes to ⟶

Lingual


Safety
Non-astronautics
Minimalism
Do only what's needed.
Consistency
Fractalness
Compactness
Orthogonality
Brevity
Grooviness
Memorability
Encapsulation
Modularity
Small interfaces between parts
Plain Data
Decoupling
Composability
Flexibility
Key: — contributes to ⟶


Lingual
Mathematical
Safety
Non-
astronautics
Encapsulation
Consistency
Minimalism
Do only what's
needed.
Plain Data
Modularity
Small interfaces
between parts
Fractalness
Compactness
Orthogonality
Decoupling
Brevity
Composability
Grooviness
Memorability
Flexibility
Key: — contributes to ⟶


Lingual
Mathematical
Safety
Non-
astronautics
Encapsulation
Consistency
Minimalism
Do only what's
needed.
Plain Data
Modularity
Small interfaces
between parts
Fractalness
Compactness
Orthogonality
Decoupling
Brevity
Composability
Grooviness
Memorability
Flexibility
Key: — contributes to ⟶


Lingual
Mathematical
Safety
Non-
astronautics
Encapsulation
Consistency
Minimalism
Do only what's
needed.
Plain Data
Modularity
Small interfaces
between parts
Fractalness
Compactness
Orthogonality
Decoupling
Brevity
Composability
Grooviness
Memorability
Flexibility
Key: — contributes to ⟶

Thank You

twitter: ErikRose

www.grinchcentral.com

erik@mozilla.com